





Re: a dozen cpu's on a chip

point  
operations.

Right.  
Amybe a  
few cpu's  
would have  
serious  
floating  
point  
power, or a  
few  
separate fp  
engines  
could be  
assigned to  
cpu's as  
needed.  
Lots of  
cpu's, doing  
stuff like  
file i/o or  
serial stuff,  
could be  
less  
powerful. I  
suppose  
we'll always  
need special  
graphics  
hardware,  
but  
just a few of  
those per  
chip.

Asymmetric  
multiprocessing makes the  
scheduler's life more  
complicated. Since the  
scheduler is part of the OS,  
and the OS is  
most often M\$, this isn't a  
good idea, IMO. ;-)  
Hardware is cheap  
(so cheap PowerPC is  
including decimal FPUs).  
Throw the FPU on  
every node, whether its

Re: a dozen cpu's on a chip

needed or not.

It  
also  
would  
make  
sense  
to  
do  
things  
like  
memory  
moves  
in  
the  
"Memory  
Mismanagement  
Unit"  
since  
the  
values  
don't  
need  
to  
be  
modified  
on  
the  
way  
through.

This  
will  
make  
it  
a  
lot  
harder  
to  
say  
how  
many  
CPUs  
are  
in  
a  
chip.  
If  
there  
is

Re: a dozen cpu's on a chip

only  
as  
much  
hardware  
as  
200  
full  
CPUs  
but  
500  
threads  
can  
be  
running  
at  
the  
same  
time,  
do  
you  
call  
it  
200  
or  
500  
CPUs.

Next step is  
to get rid of  
task  
swapping  
and threads  
altogether.  
One  
CPU is the  
OS, and one  
cpu gets  
assigned per  
process.

Which negates what you say  
above. Running a task, then  
getting an  
exception because you don't  
have an instruction you  
thought you had  
is expensive.

Re: a dozen cpu's on a chip

Why would you get an exception? If a device driver doesn't need fp opcodes, run it on one of the many cpu's that doesn't have floating point. And vice versa. <> rocket science.

You're making your scheduler's job more difficult and limiting flexibility. Computer architecture is rocket surgery.

A bunch of cpu's don't need scheduling like a single-processor os does; individual cpu's do their thing concurrently and set semaphores, and go idle, if they finish whatever they are assigned to do.

ONLY if the CPUs have tasks assigned when the system is designed. IOW, this may work for embedded processors but not general purpose computing.

And besides, the task manager cpu doesn't have anything else to do. The scheduler will mostly set up things like memory management and privileges and assignments and turn them loose, rather than frantically swapping them in real time. When everything runs simultaneously, priorities become less important. It's a whole new way of thinking.

No, it's really not new, rather rejected.

The IBM Cell chip is an architecture that trends in that direction.

Now you sound like Dimbulb. ;-) Note that the Cell processor is essentially an embedded processor. The tasks it has been designed for are quite limited.

I said it "trends in that direction", not that it was the ultimate architecture. But why is one PPC plus six simpler integer processors "quite limited"? It's obviously more general than the PPC alone.

Re: a dozen cpu's on a chip

Current hardware and software has been driven by Intel's silicon process skill (and their vicious lack of ethics) and by Microsoft's thousands of programmers (and their vicious lack of ethics) but not by any particularly intelligent planning. Most big software apps are spinning-out-of-control crapware with gigabyte service packs just pushing the bugs around. It's time for a change, for the next generation of computing, and I think it will happen when there are so many processors on a chip that multitasking quits making sense.

No, you have it backwards. Intel has been driven by hardware for the past thirty years.

I believe that's just what I said. They have just pushed an 8008 architecture – a dog when it was new – into nanometer silicon. Their attempts at cutting over to more modern architectures – iapx32, Arm, Itanic – have been expensive failures.

As many processors as we're likely to see,

there will always be more tasks/threads.

Why? What would a desktop PC need with 1024 threads?

A new language wouldn't hurt either.

Well, we can certainly agree there, though not likely on what. ;-)

Something more like Cobol, where programmers are forced to deal with the application, rather than using the application as a platform to show off how tricky they can be. Something without pointers. Something that is impossible to crash.

John

.